

## Reasoning about Loops, Specifications

## Announcements

- HW1 due Friday
  - To submit answers, first commit to SVN, then submit through Homework 1 in Homework Server
- If you have questions, please email [csci2600@cs.lists.rpi.edu](mailto:csci2600@cs.lists.rpi.edu)

2

## Outline

- Quiz 1
- Reasoning about loops (conclusion)
- Dafny lab (optional, 2pts extra towards HW2)
- Specifications

Fall 15 CSCI 2600, A Milanova

3

## Quiz 1, String Comparison

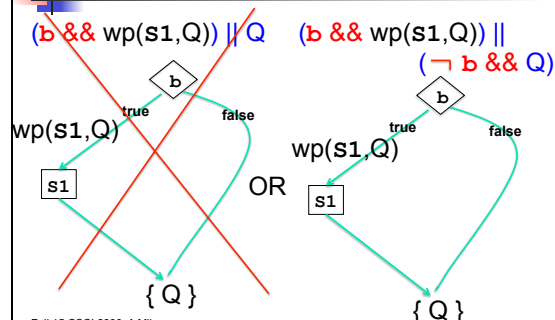
```
String a = new String("car");
String b = new String("car");
System.out.println(a == b);
// Yields false
```

```
String a = "car";
String b = "car";
System.out.println(a == b);
// Yields true!
```

Fall 15 CSCI 2600, A Milanova

4

## Quiz 1, $\text{wp}(\text{"if (b) s1;"}, Q)$



Fall 15 CSCI 2600, A Milanova

5

## So Far

- We discussed reasoning about code
  - Forward reasoning and backward reasoning
- Hoare Logic
  - Hoare Triples
  - Rule for **assignment**
  - Rule for **sequence**
  - Rule for **if-then-else**
  - Rule for **method call**
  - **Reasoning about loops**

Spring 15 CSCI 2600, A Milanova

6

## Reasoning About Loops

total correctness = partial correctness + termination

### 1. Partial correctness

- “Guess”, then prove **loop invariant**
- Loop invariant** and **loop exit condition** must imply the postcondition
- This gives us: “If the loop terminated then the postcondition holds”. But does the loop terminate?

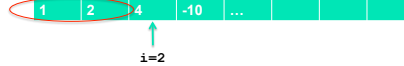
### 2. Termination

- (Informally) “Guess” **decrementing function** D. Each iteration decrements D, D at 0 along with the **loop invariant** must imply **loop exit condition**

7

## A “tedious” Invariant

Inv:  $\text{sum} = a[0] + a[1]$



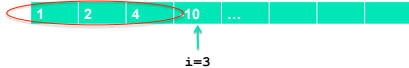
Precondition:  $\text{len} \geq 0 \ \&\& \ a.\text{length} = \text{len}$

```
int sum = 0;
int i = 0;
while (i < len)
    invariant sum = a[0] + ... + a[i-1] && i <= len
{
    sum = sum + a[i];
    i = i+1;
}
Postcondition: sum = a[0] + ... + a[a.length-1]
```

8

## A “tedious” Invariant

Inv:  $\text{sum} = a[0] + a[1] + a[2]$



Precondition:  $\text{len} \geq 0 \ \&\& \ a.\text{length} = \text{len}$

```
int sum = 0;
int i = 0;
while (i < len)
    invariant sum = a[0] + ... + a[i-1] && i <= len
{
    sum = sum + a[i];
    i = i+1;
}
Postcondition: sum = a[0] + ... + a[a.length-1]
```

9

## A “tedious” Invariant

Inv:  $\text{sum} = a[0] + a[1] + \dots + a[a.\text{length}-1]$



As the induction variable  $i$  moves through the array,  $\text{sum}$  contains the correct partial result.

At the end of the array,  $\text{sum}$  contains the correct total

Similar invariants for min, max, avg, sorting algorithms, etc. These invariants are easy to guess.

Fall 15 CSCI 2600, A Milanova

10

## Let's Catch the Bug

Precondition:  $\text{len} \geq 0 \ \&\& \ a.\text{length} = \text{len}$

```
int sum = 0;
int i = 0;
while (i <= len)
    invariant sum = a[0] + ... + a[i-1] && i <= len+1
{
    sum = sum + a[i];
    i = i+1;
}
Postcondition: sum = a[0] + ... + a[a.length-1]
```

Fall 15 CSCI 2600, A Milanova

11

## “Interesting” Invariant

### Another Factorial

Precondition:  $n \geq 0$

```
r = 1;
n = t;
while (n != 0) {
    r = r*n;
    n = n-1;
}
```

Postcondition:  $r = t!$

Fall 15 CSCI 2600, A Milanova (example due to Michael Ernst, UW)

12

## Interesting Invariant

### Integer Division

Precondition:  $x \geq 0 \ \&\& \ y > 0$

```
r = x;
q = 0;
while (y <= r) {
  r = r-y;
  q = q+1;
}
Postcondition:  $x = y*q + r \ \&\& \ r < y$ 
```

Fall 15 CSCI 2600, A.Milanova (example due to Michael Ernst, UW)

13

## Interesting Invariant

Precondition:  $x1 > 0 \ \&\& \ x2 > 0$

```
y1 = x1;
y2 = x2;
while (y1 != y2)
  invariant gcd(y1,y2) = gcd(x1,x2)
{
  if (y1 > y2) {
    y1 = y1-y2;
  } else {
    y2 = y2-y1;
  }
}
Postcondition:  $y1 = \text{gcd}(x1,x2)$ 
```

14

## Dafny Lab

You may have to add a Dafny function to state the postcondition. E.g., pow2(n):

```
function pow2(n: nat): nat
{ if n == 0 then 1 else 2*pow2(n-1) }
```

### Trailing Zeros

Precondition:  $x \neq 0$

```
zeros = 0;
y = x;
```

1. Write TrailingZeros in Dafny
2. Verify your program with Dafny
3. Cut and paste your program into file DafnyLoops.txt, then submit it to the Homework Server, Principles of Software Lab 2, Deadline midnight Wednesday.

```
while (y%10 == 0) {
  y = y/10;
  zeros = zeros + 1;
}
Postcondition:  $x = y*10^{\text{zeros}} \ \&\& \ y\%10 \neq 0$ 
```

15

## Outline

- Specifications
  - Benefits of specifications
  - Specification conventions
    - Javadoc
    - PoS specifications
    - JML
  - Specification style
  - Specification strength

16

## Specifications

- A specification consists of a **precondition** and a **postcondition**
  - Precondition: conditions that hold before method executes
  - Postcondition: conditions that hold after method finished execution (if precondition held!)

17

## Specifications

- A specification is a **contract** between a method and its caller
  - Obligations of the method (**implementation** of the specification): agrees to provide postcondition **if precondition held**
  - Obligations of the caller (**user** of specification): agrees to meet the precondition and not expect more than the promised postcondition
- A specification is an **abstraction**

18

## Example Specification

Precondition: `len ≥ 0 && a.length = len`

Postcondition: `result = a[0]+...+a[a.length-1]`

```
int sum(int[] a, int len) {
    int sum = 0;
    int i = 0;
    while (i < len) {
        sum = sum + a[i];
        i = i+1;
    }
    return sum;
}
```

For our purposes, we will be writing specifications that are a bit less formal than this example. Mathematical rigor is welcome, but not always necessary.

Fall 15 CSCI 2600, A Milanova

19

## Benefits of Specifications

- Precisely documents method behavior
  - Imagine if you had to read the code of the Java libraries to figure what they do!
- Promotes **modularity**
  - Modularity is key to the development of correct and maintainable software
  - Specifications help organize code into small **modules**, with clear-cut boundaries between those modules

Fall 15 CSCI 2600, A Milanova

20

## Benefits of Specifications

- Promote modularity...
  - Client relies on description in specification, no need to know the implementation
  - Method must support specification, but its implementation is free otherwise!
  - Method and client can be built **simultaneously**
- Enables reasoning about correctness
  - “Does code do the right thing?” means “Does code conform to its specification?”
  - Confirmed by testing and **verification**

21

## So, Why Not Just Read Code?

```
boolean sub(List<T> src, List<T> part) {
    int part_index = 0;
    for (Object o : src) {
        if (o.equals(part.get(part_index))) {
            part_index++;
            if (part_index == part.size()) {
                return true;
            }
        } else {
            part_index = 0;
        }
    }
    return false;
}
```

Fall 15 CSCI 2600, A Milanova (example due to Michael Ernst, UW)

22

## So, Why Not Just Read Code?

- Code is complicated
  - Gives a lot more detail than client needs
  - Understanding code is a burden
- Code is ambiguous
  - What is **essential** and what is **incidental** (e.g., result of an optimization)
- Client needs to know **what** the code does, not **how** it does it!

Fall 15 CSCI 2600, A Milanova (based on slides by Michael Ernst, UW)

23

## What About Comments?

```
// method checks if part appears as
// subsequence in src
boolean sub(List<T> src, List<T> part) {
    ...
}
```

Comments are important, but insufficient. They are informal and often ambiguous

Fall 15 CSCI 2600, A Milanova

24

## Specifications Are Concise and Precise

- Unfortunately, lots of code lacks specification ☹
- Programmers guess what code does by reading the code or running it
- This results in bugs and/or complex code with unclear behavior
- Specification inference is an active area of research

Fall 15 CSCI 2600, A Milanova (based on slide by Michael Ernst)

25

## So, What's in `sub`'s Specification?

```
boolean sub(List<T> src, List<T> part) {  
    int part_index = 0;  
    for (Object o : src) {  
        if (o.equals(part.get(part_index))) {  
            part_index++;  
            if (part_index == part.size()) {  
                return true;  
            }  
        } else {  
            part_index = 0;  
        }  
    }  
    return false;  
}
```

Fall 15 CSCI 2600, A Milanova (example due to Michael Ernst, UW)

26

## So, What's in `sub`'s Specification?

Choice 1:

// `sub` returns true if `part` is a subsequence of `src`; it returns false otherwise.

Choice 2:

// `src` must be non-null

// If `src` is the empty list, then `sub` returns false

// Requirements on `part` too...

// If there is a partial match in the beginning, `sub` returns false, even if there is a full match later.

E.g. `sub([1, 2, 1, 2, 1, 3], [1, 2, 1, 3])` is false.

27

## What's in `sub`'s Specification?

- This complex specification is a red flag
- Rule: **it is better to simplify design and code rather than try to describe complex behavior!**
- If you end up writing a complicated specification, redesign and rewrite your code --- either something is wrong or code is extremely complex and hard to reason about (Mike Ernst calls this "a sneaky fringe benefit of specifications")

Fall 15 CSCI 2600, A Milanova (based on slides by Michael Ernst)

28

## Goal of Principles of Software

- One goal of Principles of Software is to instill a discipline of writing concise and precise specifications

Fall 15 CSCI 2600, A Milanova

29

## Outline

- Specifications
  - Benefits of specifications
  - **Specification conventions**
    - Javadoc
    - PoS specifications
    - JML
  - Specification style
  - Specification strength

30